

SDEV1201

LESSON 2

ERDS AND SQL INTRODUCTION

Let's review

Data vs. Information

Data: Data refers to raw, unorganized facts or figures that can be collected and stored, and it can be in the form of numbers, text, images, or any other type of input. Data, by itself, lacks context and meaning. It is the most basic form and requires further processing to become useful.

Information: Information is the processed and organized form of data. It is data that has been analyzed, structured, and given context. Information provides meaning and can be used to answer questions or make decisions.

If you have the numbers “13, 15, 18”; it is just data. However, if you put it into context: “The temperatures over the last three days are 13, 15, and 18” it becomes information because it provides meaning and context (i.e. what is the current temperature trend).

Knowledge

Knowledge goes beyond information in that it involves understanding and expertise. It is the result of gaining insights, experience, and being able to apply information in a meaningful way.

Knowledge is the culmination of information and personal understanding, allowing individuals to make informed judgments and take effective action.

What is a database?

Databases are an organized collection of data. There are many kinds of databases.

In SDEV1201 we will be using a type of database called a relational database. A relational database is where like information is stored in different tables that all relate to each other in some way

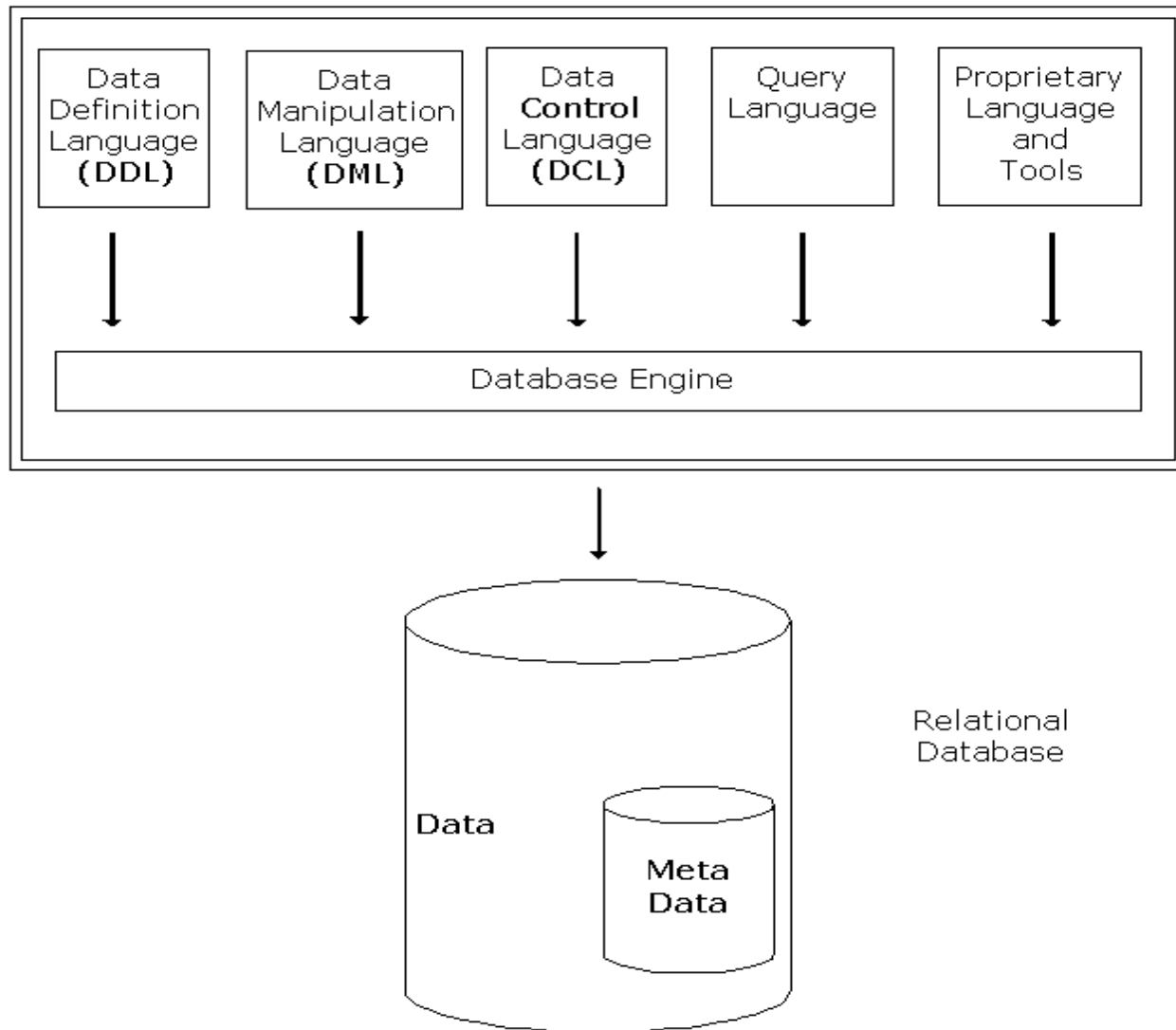
Databases are necessary for large organizations for the following reasons:

- Size
- Ease of updating
- Accuracy
- Security
- Redundancy
- Importance

Databases and DBMS

Databases are what store the data. To effectively maintain the data we use a Database Management System (DBMS). You would use the DBMS to manage your databases. The database is the data and the rules. The DBMS allows you to interact with your db.

Database Management System (DBMS)



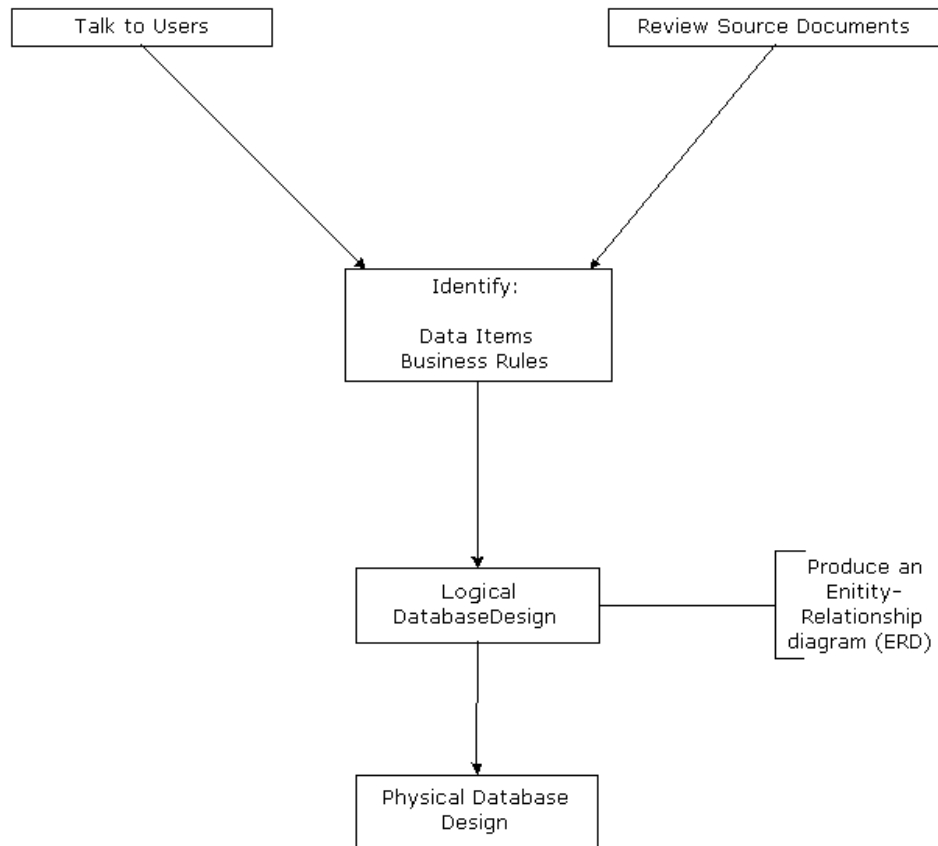
The database engine is the core set of programs that make up a DBMS. These programs execute when a request is submitted to the DBMS and directly interact with the database itself to provide a service. For example, a user adds new data to the database. The user would execute an application program to perform this task. The application program would submit a DML statement to the DBMS requesting that new data be added to the database. The DML statement causes one or more programs in the database engine to execute and actually add the new data to the database. It is important to note that developers and users do not interact directly with the database; instead they submit requests to the DBMS. The DBMS responds to requests from the various stakeholders in the system (e.g. users or developer) and performs the various tasks involved in creating, changing, and using a database.

The Relational DBMS

A relational DBMS uses tables to store data in the database. A table consists of rows and columns (similar to a spreadsheet). For example, information about customers would be stored in a table named Customer in the database as shown below:

<u>CustomerNumber</u>	FirstName	<u>LastName</u>	Address	Phone
100	Adams	Julie	9825-77 Ave Edmonton AB T5C 8E2	780-111-1111
200	Smith	Miles	4308-133 Ave Edmonton AB T5C 8E9	780-222-2222

The Database Design Process



The database design process is a methodology used to design the structure of a database for an application. Prior to starting the database design process the designer must understand:

- The purpose of the database (i.e. Why does one need the database?).
- The data that will be stored in the database.
- What functions the database will support (e.g. add a customer record).

The database designer develops this understanding by completing systems analysis and design tasks such as:

- Talking to the users to identify requirements.
- Reviewing source documents (e.g. reports, screen layouts, paper-based business forms).

Create a logical database design

CustomerNumber	LastName	FirstName	Phone
1	Jones	Abby	439 - 1763
2	Davis	James	485 - 4309

The next step is to create a logical database design. A logical database design represents the programmer's and the user's perception of the corporate data stored in the database. For example, when using a relational database management system, data is organized into a collection of tables (rows and columns) regardless of the actual arrangement of the data on the physical storage device (e.g. a hard-drive). A Customer table is used to store data about customers as depicted above.

Each row in the table represents one specific customer. Each column in the table represents one attribute or characteristic of a customer.

Normalization Overview

Normalization is a set of guidelines for creating the logical design for a relational database. Normalization helps database designers decide:

- The number of entities/tables required within the database.
- Which attributes/columns belong in which entities/tables.

The normalization process helps minimize redundant data within the database and helps ensure the data remains consistent. An example of inconsistent data would be having two tables in a database that record customer information, and having conflicting data (e.g. phone number), for the same customer, stored in the two tables. Using the normalization process to fine tune the design produces database designs that minimize the occurrence of inconsistencies within the data. The logical design process will be expanded in later lessons.

Physical Design

When the logical design is complete a physical design for the database will be created. The physical design describes the physical structure of the data. The physical design describes how the logical design will be implemented. Introductory topics related to physical design will be discussed later in the course. More advanced physical design topics to improve performance such as database optimization and tuning, are beyond the scope of this course.

Dialects of SQL

PL/SQL — Found in Oracle. PL/SQL, which stands for Procedural Language/SQL and contains many similarities to the general programming language Ada; IBM DB2 added (limited) support for Oracle's PL/SQL in version 9.5.

Transact-SQL — Used by both Microsoft SQL Server and Sybase Adaptive Server. As Microsoft and Sybase have moved away from the common platform they shared early in the 1990s, their implementations of Transact-SQL have also diverged, producing two distinct dialects of Transact-SQL.

SQL PL — IBM DB2's procedural extension for SQL, introduced in version 7.0, provides constructs necessary for implementing control flow logic around traditional SQL queries and operations.

PL/pgSQL — The name of the SQL dialect and extensions implemented in PostgreSQL. The acronym stands for Procedural Language/postgreSQL.

MySQL — MySQL has introduced a procedural language into its database in version 5, but there is no official name for it.

ACID database

An ACID database is a relational or distributed database that adheres to the ACID properties (Atomicity, Consistency, Isolation, Durability), which guarantee that database transactions are processed reliably and correctly, even in the event of errors, power failures, or other system mishaps. These properties ensure that data remains valid and maintains its integrity by treating transactions as a single, indivisible unit.

Today's Agenda

- ❑ Entity Relationship Diagrams
- ❑ Describe entities and attributes
- ❑ Introducing SQL

Entity Relationship Model

The logical database design process uses the entity-relationship model to represent the business scenario as a collection of entities and relationships. The entity-relationship diagram (ERD) documents the logical design and becomes a starting point for the physical design process, which defines the structure of the database. This lesson will discuss concepts related to the entity-relationship model.

The **Entity-Relationship** model is a tool that organizes and documents the logical design of a database. The entity-relationship model describes the data used by an application in terms of **entities, attributes, and relationships**.

Note: For reference see the “2A Entity Relationship Model” Word document located in the “Week 2 – ERDs” section of Brightspace.

Entity Relationship Model

Entity

An entity is a person, place, thing or concept that is used in the business context of the application and for which data is collected.

Order Number: 5

Date: October 10, 2000

Nuts n Bolts Hardware

12538 - 77 Ave
Edmonton, Alberta
T6E 9K7

Ph: 436 - 9812 Fax: 433 - 5187

Customer Information

Number: 100

Joan Adams
9825 - 77 Ave
Edmonton, Alberta
T5C 8E2

PH: 488 - 8712

What entities are in this view?
The entities involved in this
information include: Order,
Customer and Item.

Item	Description	Quantity	Price	Amount
H-101	Widgets	10	1.00	10.00
H-229	Bolts	5	0.50	2.50
Subtotal				12.50
GST				.88
Total				13.38

Entity

Entities can be physical things that you can touch (e.g. a customer) or conceptual things that do not physically exist (e.g. a course). It is important to distinguish between an **entity** and an **instance of an entity**. An instance of an entity is one specific occurrence of an entity. In the order example Customer is an entity. Joan Adams is an instance of the customer entity. An entity can have many instances (hopefully the hardware business is booming and there are many customers!).

Attributes

A piece of data that describes an entity is called an **attribute**. Synonyms for attribute include: element, property and field. In the order example, the customer entity has the following attributes:

- Customer Number
- Last Name
- First Name
- Address
- Phone Number

The values for all attributes describing an entity make up one instance of the entity. For example, the following values describe one specific customer.

Customer Number	Last Name	First Name	Address	Phone
100	Adams	Joan	9825-77 Ave Edm Ab T5C 8E2	488-8712

Types of Attributes

There are two types of attributes: **atomic** and **composite**. An atomic attribute has a value that is in its simplest form. Atomic attributes cannot be subdivided into two or more other meaningful attributes. "Customer Number" is an example of an atomic attribute. Generally speaking, storing data in atomic form adds flexibility to your application. A composite attribute is an attribute that can be divided into two or more atomic attributes. "Address" is an example of a composite attribute. Address can be broken down to street address, city, province, and postal code; all of which are atomic attributes.

Types of Attributes

Business rules play a role in deciding how to store data (atomic vs. composite). Consider the "Address" attribute. For a national company all addresses have the same format. Therefore, address is stored in its atomic form. For an international company addresses vary in format (Americans use a five digit zip code while Canadians use a six digit postal code). In this situation address would be stored in its composite form as a single attribute.

Types of Attributes

There are 2 types of attributes: **atomic** and **composite**.

e.g. in the hardware store example, the customer number is atomic and the address is composite because it could be divided into street address, city, province, postal code, etc.

Classifying Attributes

Attributes can be classified as **stored** or **derived**. A stored attribute has its value physically stored in the database.

e.g. An order number is stored because the value is physically stored in the database.

The order amount could be a derived attribute because it is calculated by multiplying Quantity and Price.

Values an Attribute Can Assume

Every attribute has a set of legitimate values, defined in terms of the **data type** and **domain** of the attribute. The **domain** is the set of values that an attribute can assume.

*e.g. the customer number must be a **number greater than zero***

Special Cases: The **NULL** value. **NULL** either means:

1. The value is unknown. Consider creating an instance of the customer entity. If the customer does not know their postal code the null value can be stored as the postal code for the new record. The customer can supply their postal code at a later date. The record can then be updated to reflect the correct postal code.
2. The attribute does not apply for this particular instance. Consider an employee entity. One attribute could be the name of the employee's spouse. The null value would be recorded in the spousal name attribute for all records that represent a single employee.

Note: The null value must be used with caution, as it propagates through expressions. This means that any expression using an attribute that contains the null value will result in null.

Primary Key

Many tasks within an application involve using one specific instance of an entity. For example, changing the price of a specific item in inventory. To accomplish this, the application must have a way of uniquely identifying each instance of the item entity. The primary key allows the DBMS to uniquely identify each instance of an entity. The primary key is an attribute, or group of attributes, that has a unique value for each instance of an entity.

e.g. In the Customer Order example, the Item Number uniquely identifies each item in inventory, and can act as the Primary Key for the Item entity.

e.g. your student ID uniquely identifies you as a student at NAIT: no other student has the same student ID.

The bottom line is that each entity must have a primary key that enables the DBMS to identify each instance of the entity.

Primary Keys

Entities which don't have a naturally occurring unique attribute may have one made up by the database designer. These are called **technical keys**.

A combination of two or more attributes acting as the primary key is called a **concatenated key** or **composite key**.

StudentID	Lastame	FirstName	Address	City	Province
1	Chen	James	123 Candy Cane Lane	Edmonton	AB
2	Henke	Greg	33 Willow Dr.	Edmonton	AB
3	King	Mike	77 Main St.	Lumsden	SK

Relationships

A relationship represents a business association between two entities. Relationships are based on the business rules of the organization. All relationships are bi-directional in that they can be interpreted from the point of view of each entity. In the customer order example a relationship exists between the customer entity and the order entity. This reflects the fact that each time a customer makes a purchase an order is created. From the customer perspective, one customer can place many orders. From the order perspective, one and only one customer places an order.

Relationships are categorized based on how the instances of the entities are related to each other. The relating of instances between entities is called **cardinality**. The three major types of cardinality are:

Types of Relationships

- One-to-one

- These are very rare.
- *e.g. each country has **one UN representative**, and each UN representative represents only **one country**. The Country entity and the UNRepresentative entity have a 1:1 relationship.*

- One-to-many

- We'll see these LOTS.
- *e.g. each Department has **many employees**, but each employee belongs to a **single department**. The Employee entity and the Department entity have a 1:many relationship.*

- Many-to-many

- We'll see these LOTS, and we'll do something special in our design to represent them.
- *e.g. a student enrolls in **many classes**, and each class has **many students**. The Student entity and the Class entity have a many:many relationship.*

Symbols for Cardinality

Here are the symbols associated with the crow's foot notation:

Zero



The symbol/diagram above denotes zero in crow's foot notation. We know this because of the zero/circle indicator at the right side of the horizontal line.

One



The diagram above shows a horizontal line with a short vertical line crossing it. The vertical line acts as the indicator – it denotes one.

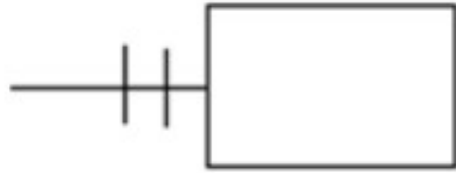
Many



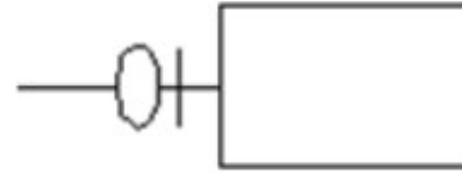
The diagram above denotes many. You can easily remember this symbol because it looks like a crow's foot.

The three diagrams are the basic representation of indicators in crow's foot notation. But in most cases, these indicators are combined to fully understand the relationship between entities.

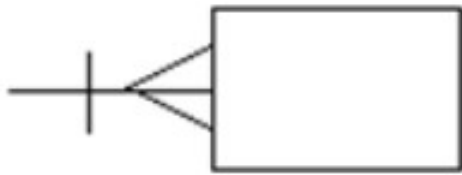
Symbols for Cardinality



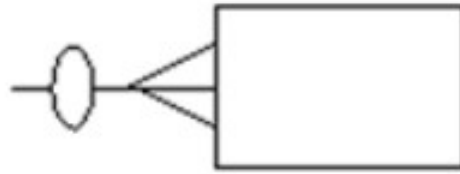
... to one
(mandatory 1)



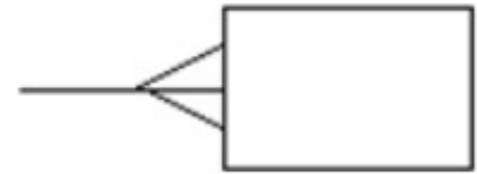
... to zero or one
(optional 1)



... to one or many
(mandatory many)



... to zero, one, or many
(optional many)



... to many

Foreign Keys

A foreign key is the mechanism that defines a relationship between entities. **A foreign key is a primary key of one entity that appears as an attribute of another entity.** In each relationship one entity has a foreign key and the other entity has the related primary key. The **entity with the foreign key is referred to as the child entity in the relationship.** The **entity with the primary key is referred to as the parent entity in the relationship.** The instance in the child entity is said to reference or refer to the related instance in the parent entity. Foreign keys are used to define all relationships between entities. The foreign key and related primary **key must be the same data, however the two attributes do not need to have the same name.** The name of the foreign key attribute should be descriptive and "make sense" with the business scenario. In some cases, the foreign key attribute will have the same name as the related primary key attribute in the parent entity; however, this is not always the case.

Foreign keys

A **foreign key** is how we define relationships between entities.

A foreign key is a **primary key of one entity** (the “parent”) that appears as an **attribute of another entity** (the “child”).

Note: the two attributes may or may not have the same name. e.g. the Student ID in the Marks table may be called just ID in the Student table.

Foreign keys – an example

A company has multiple departments.

Each department has one manager.

A manager can manage one (and only) one department within the company.

What kind of relationship exists between Department & Manager?

Which is the parent and which is the child?

This business rule translates to a one-to-one relationship between the employee entity and the department entity. This relationship can be defined by:

1. Adding the DepartmentId attribute (primary key of the Department entity) as a foreign key, named "ManagesDept", to the Employee entity.
2. Adding the EmployeeId attribute (primary key of the Employee entity) as a foreign key, named "Manager", to the Department entity.

Foreign keys – an example

Employee

EmployeeId (pk)	Last Name	First Name	Manages Dept (fk)
100	Smith	Jon	10
200	Adams	Jayne	
300	Everett	Beverly	5
400	Williams	Greg	
500	Stiles	Adam	

Department

DepartmentId (pk)	Department Name
5	Research
10	Marketing

Either method will work. Consider the first method, adding the DepartmentId to the Employee entity as a foreign key named Manages Dept.

This is not a good choice as many employees are not managers. This results in many instances in the Employee entity with the null value stored as the value for the Manages Dept attribute.

Foreign keys – an example

Employee

EmployeeId (pk)	Last Name	First Name
100	Smith	Jon
200	Adams	Jayne
300	Everett	Beverly
400	Williams	Greg
500	Stiles	Adam

Department

DepartmentId (pk)	Department Name	Manager (fk)
5	Research	300
10	Marketing	100

A better approach is to add the EmployeeId attribute to the Department entity as a foreign key named Manager.

Another example

A retail company sells merchandise. A single customer can make multiple orders. Each order is made out to one individual customer.

What is the relationship between the Customer entity and the Order entity?

Answer: One to Many

Which is the parent?

Answer: Customer

One To Many Relationship

Customer

Customer Number (pk)	Last Name	First Name	Address	City	Province	Postal Code	Phone
100	Jones	Grace	9825-77 Ave	Edmonton	Alberta	T5R 2E7	433-8126
110	Davis	Tyler	23 Apple Dr.	St. Albert	Alberta	T3Y 8B1	459-6194
120	Birney	Ann	13853 - 66 st	Edmonton	Alberta	T3R 2U8	466-8374

Order

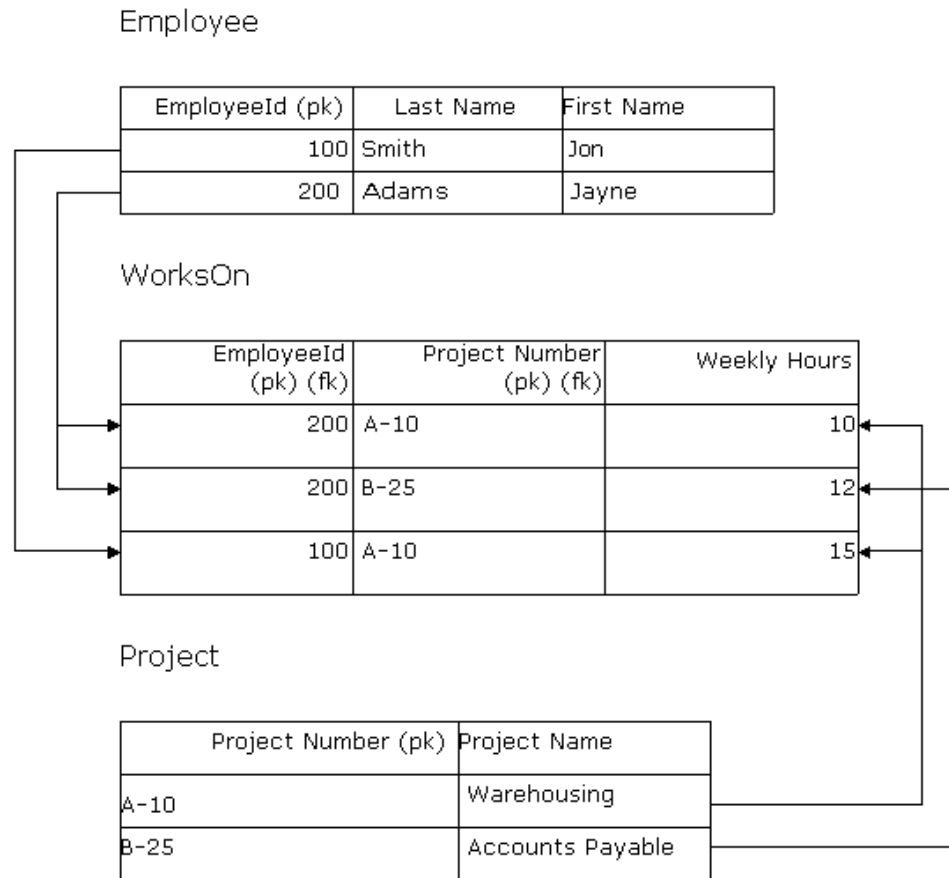
Order Number (pk)	Date	Subtotal	GST	Total	Customer Number (fk)
5	Oct 10, 2000	12.50	0.88	13.38	120
10	Oct 22, 2000	10.00	0.70	10.70	110
15	Oct 24, 2000	20.00	1.40	21.40	110



The Many-To-Many Relationship

The relational database does not facilitate a many-to-many relationship between two entities. Hence, many-to-many relationships need to be reduced to one-to-many relationships. Creating a third entity referred to as an associative entity does this. The associative entity contains the primary key attributes of the two entities that share the many-to-many relationship as well as any other attributes that describe the association between the entities. Two one-to-many relationships are defined between the associative entity and the entities that share the many-to-many relationship.

Many-To-Many relationships cont'd



Consider a consulting company. The company develops many projects simultaneously. Multiple employees make up a project team. One employee can be assigned to multiple projects simultaneously. These business rules translate to a many-to-many relationship between the Project entity and the Employee entity. The associative entity named WorksOn defines this relationship. The WorksOn entity contains the following attributes:

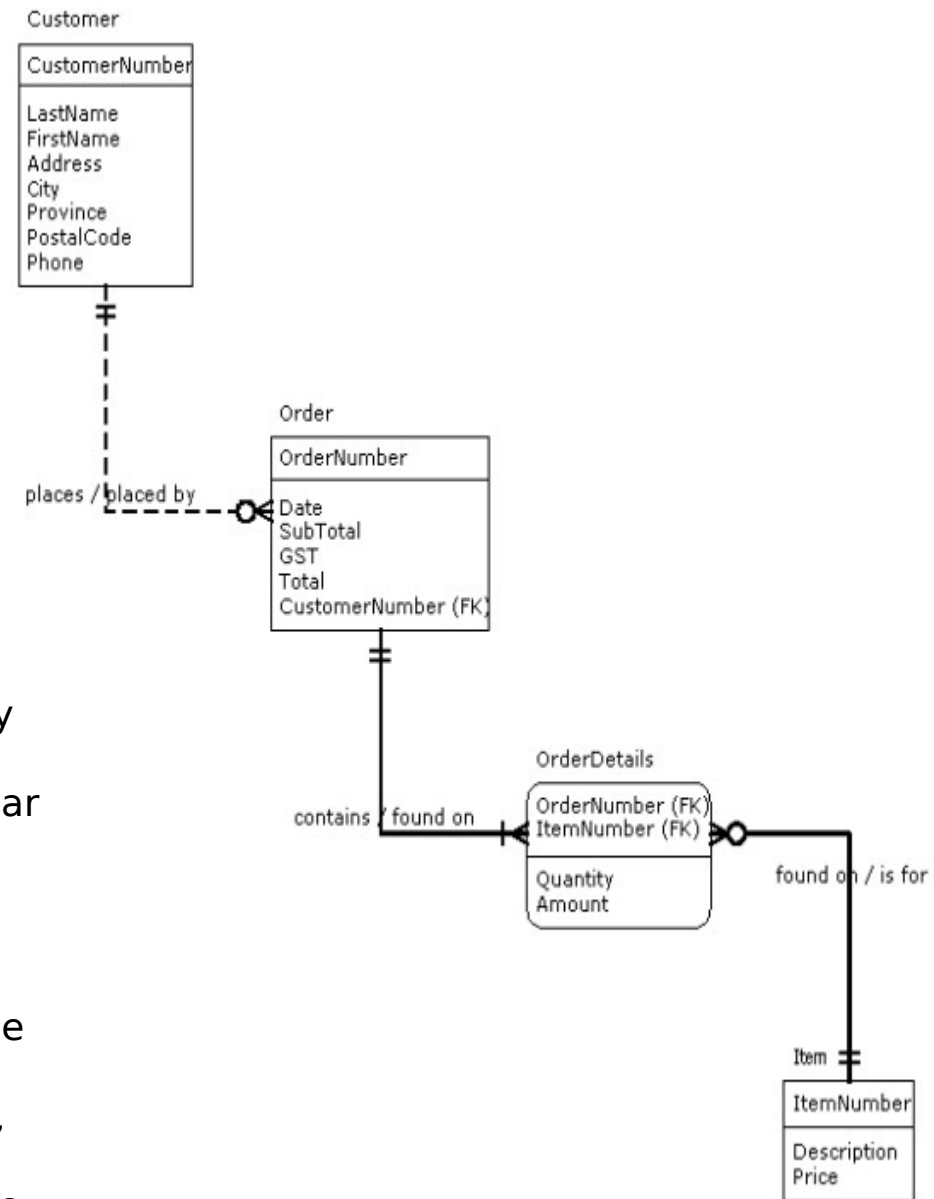
- EmployeeId (primary key of the Employee entity).
- Project Number (primary key of the Project entity).

Many-To-Many relationships cont'd

The primary key of the associative entity is always made up of two or more attributes. The primary keys from the entities that share the many-to-many relationship form the primary key for the associative entity. In this example, EmployeeId and ProjectNumber make up the primary key for the WorksOn entity.

A one-to-many relationship is defined between the Employee entity (parent) and the WorksOn entity (child). A one-to-many relationship is defined between the Project entity (parent) and the WorksOn entity (child). Note that a single attribute (e.g. EmployeeId in the WorksOn entity) can act as both a primary key and a foreign key within the same entity.

Back to our Sample ERD



A box represents an entity. If the box has square corners it is a base entity. If the box has rounded corners it is an associative entity. The name of the entity appears above the box. All attributes describing the entity are listed inside the box. Primary key attributes are listed first, and appear above the line. Attributes that do not act as the primary key for the entity appear below the line. Any attribute that acts as a foreign key are suffixed with "FK".

A line connecting two boxes represents a relationship between the two entities. The line has cardinality symbols that denote the type of relationship. A dashed line represents a non-identifying relationship. This means that the foreign key, in the child entity, does not act as part of the primary key for the entity. The relationship between the Customer entity and the Order entity is an example of a non-identifying relationship. A solid line represents an identifying relationship. This means that the foreign key does act as part of the primary key in the child entity. The relationship between the Order entity and the OrderDetails

The Entity-Relationship Diagram

Relationships are labeled with a verb phrase. This allows the relationship to be read much like a sentence. The entities are the nouns and the relationship label is the verb. Relationships can be read in two directions:

- From parent to child.
- From child to parent.

For the relationship between the Customer entity and the Order entity, Customer is the parent and Order is the child. Reading from parent to child: Customer places Order. Reading from child to parent: Order placed by Customer.

Terminology so far

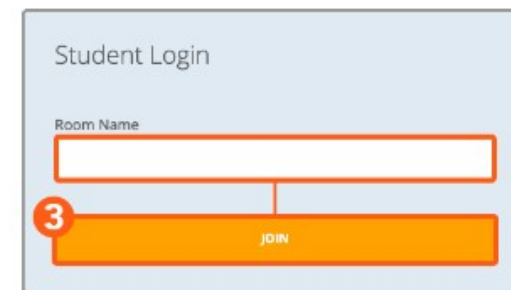
- ❑ Atomic and composite attributes
- ❑ Stored and derived attributes
- ❑ How do we define the values an attribute can hold?
- ❑ Primary key
- ❑ Technical key
- ❑ Concatenated or composite key
- ❑ Foreign key
- ❑ Parent & child relationship
- ❑ Associative entity

Break

Cardinality quiz!

socrative.com

1. Click **Login**
2. Select **Student Login**
3. Enter room **ZUBIS**



Introducing SQL

PostgreSQL Install

Open <https://www.postgresql.org/download/windows/>

Click on Download the installer. It will take you to <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>

Download PostgreSQL version 17.6. Click on the download button.

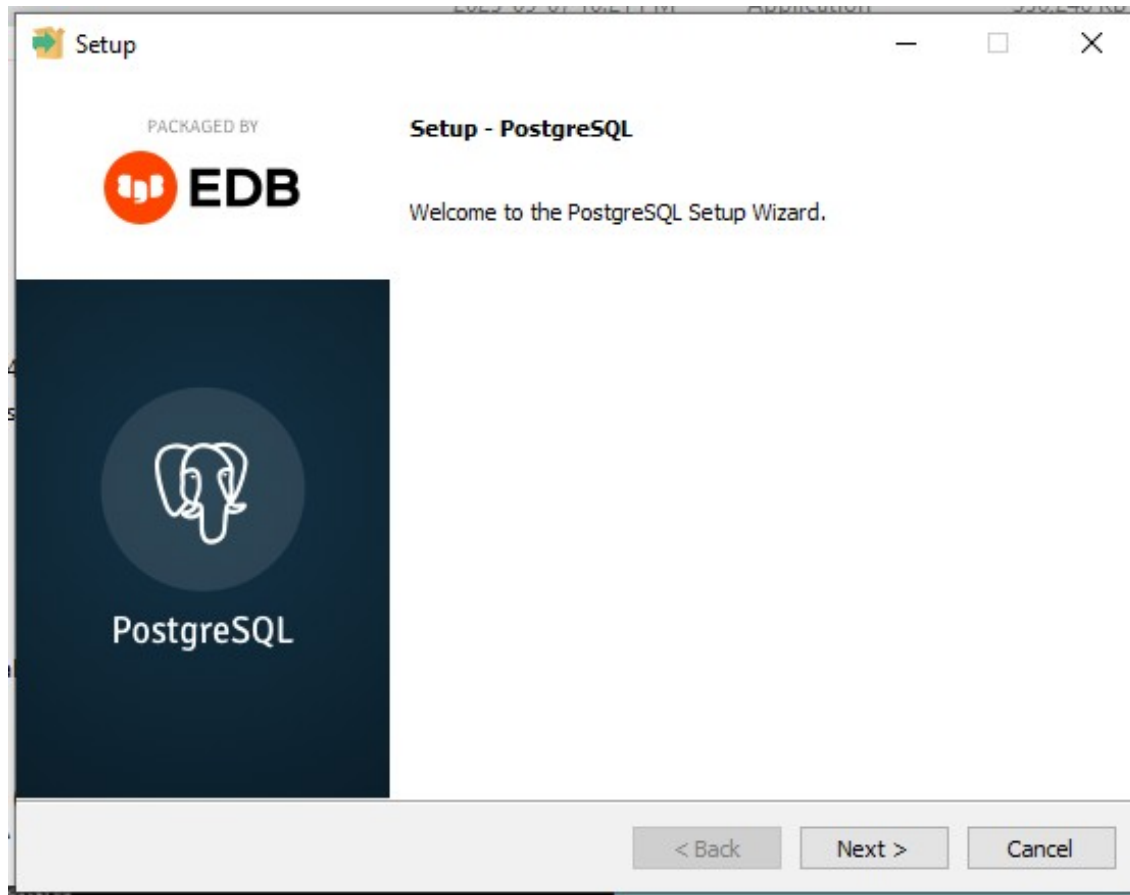
PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
17.6	postgresql.org 	postgresql.org 		 	Not supported

PostgreSQL Install

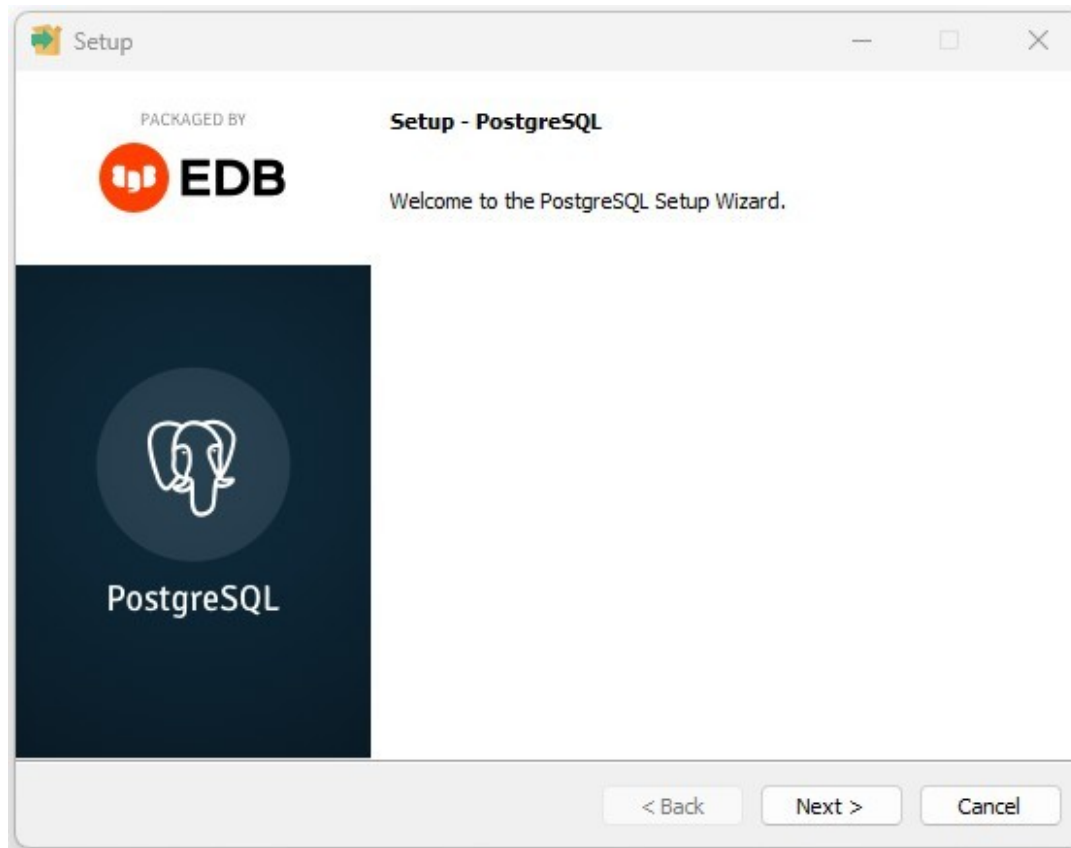
The installation file will download to your default Downloads folder. Right-click on

postgresql-17.6-1-windows-x64.exe and choose “Run as Administrator”.

PostgreSQL Install

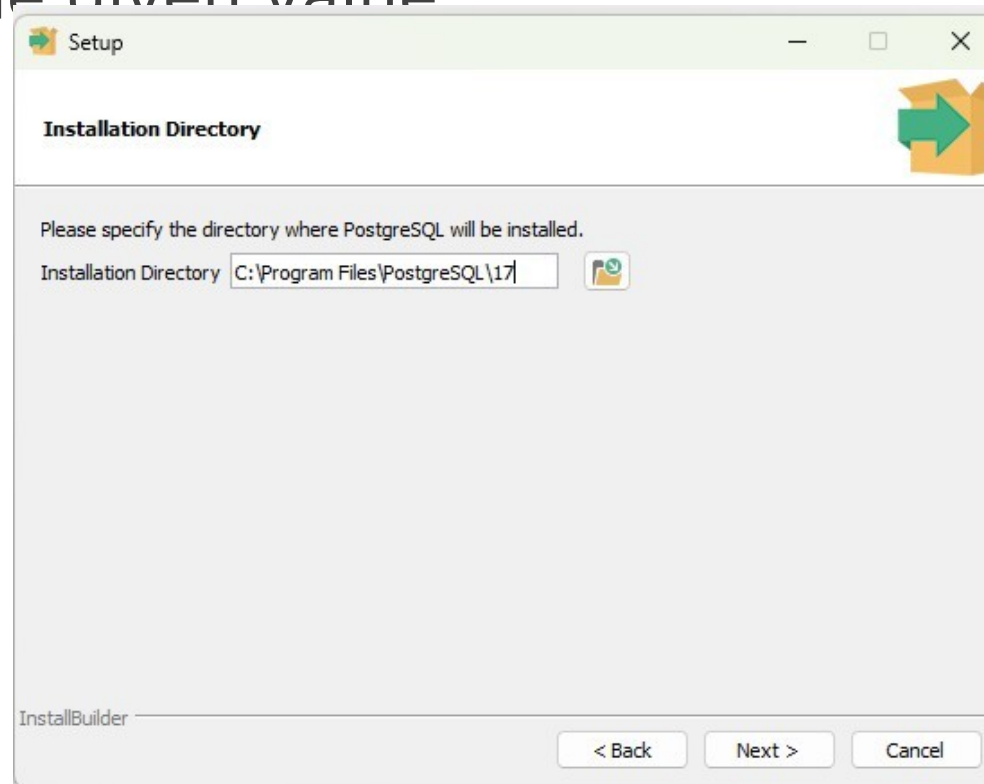


PostgreSQL Install



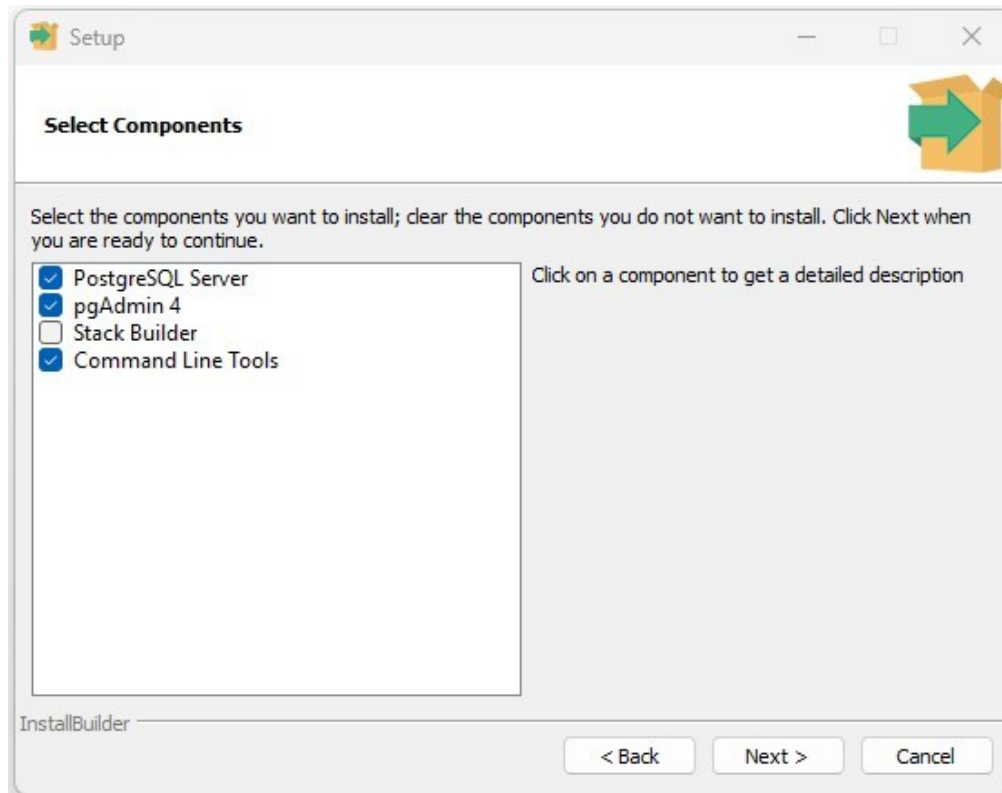
PostgreSQL Install

Leave the default unless you have a very specific reason to change the given value

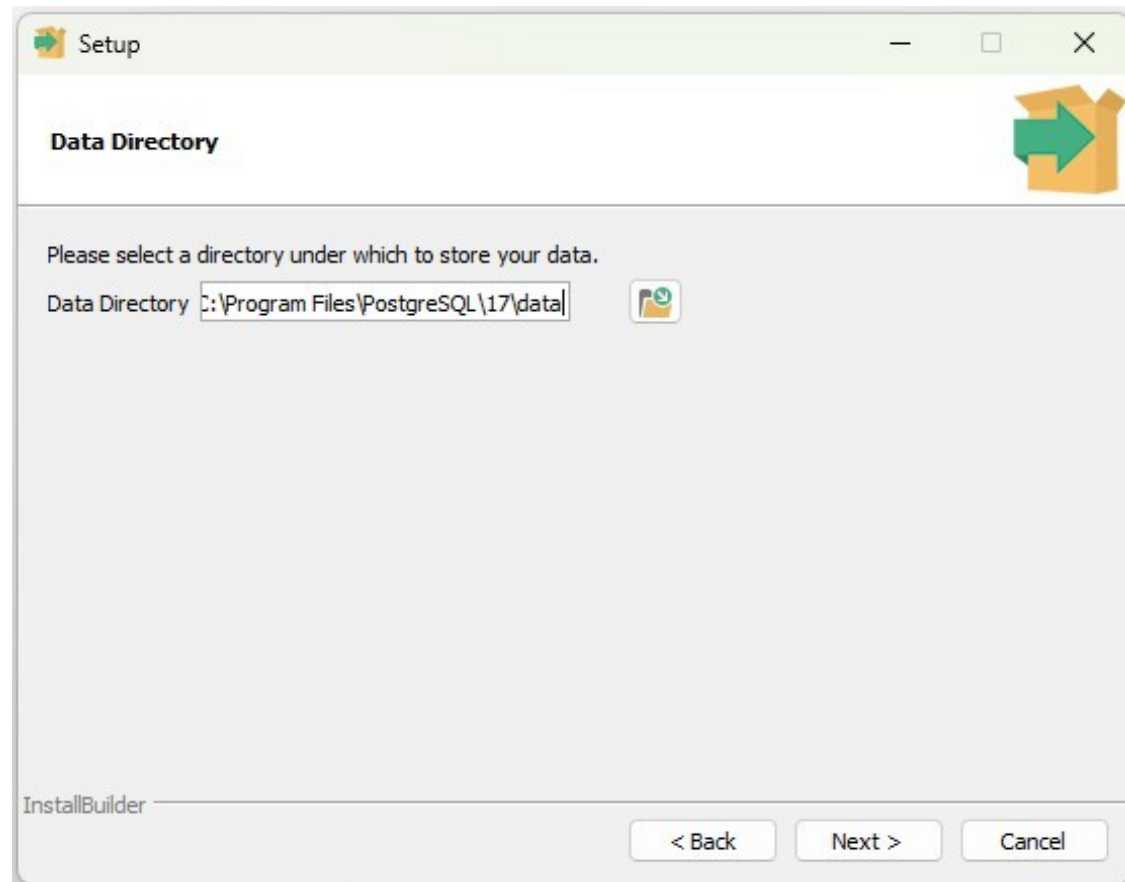


PostgreSQL Install

Uncheck Stack builder.

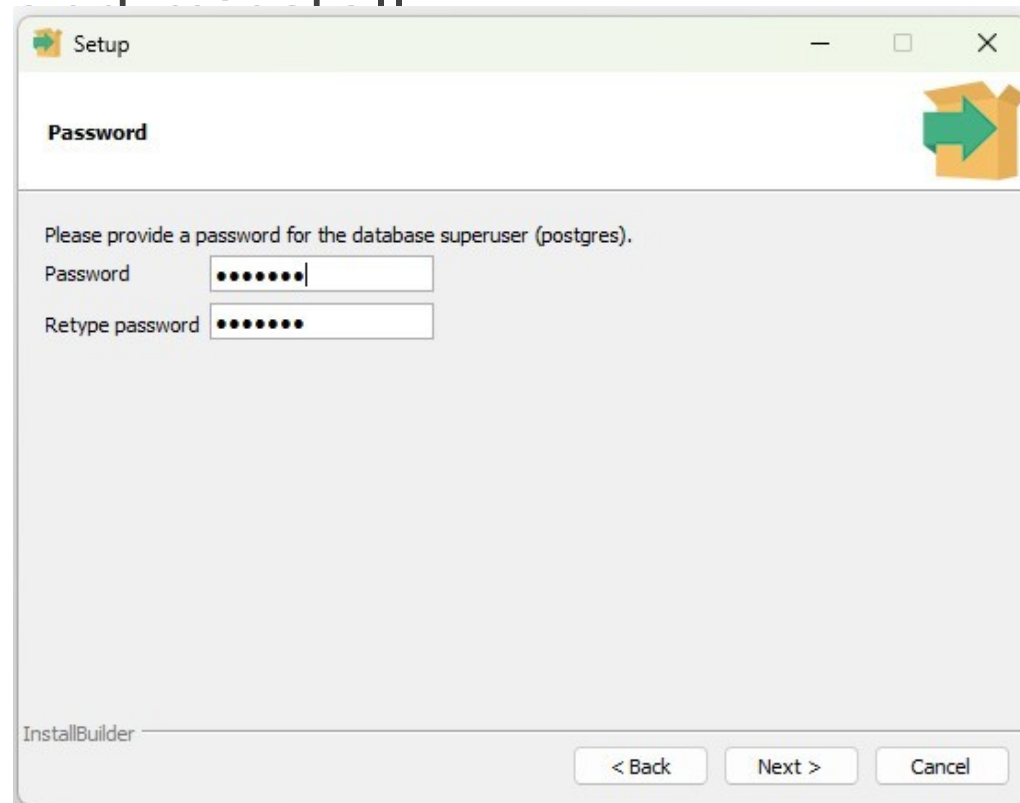


PostgreSQL Install



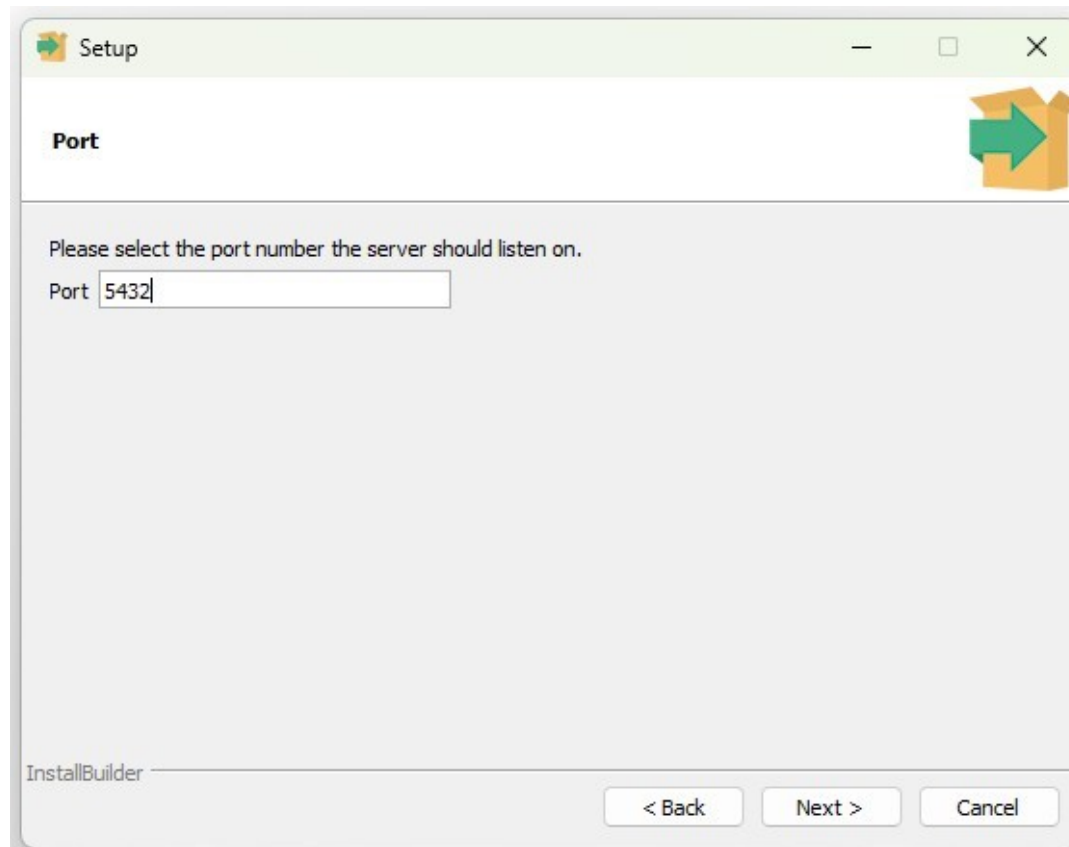
PostgreSQL Install

Make sure you keep the password safe, otherwise you will have to uninstall and reinstall.

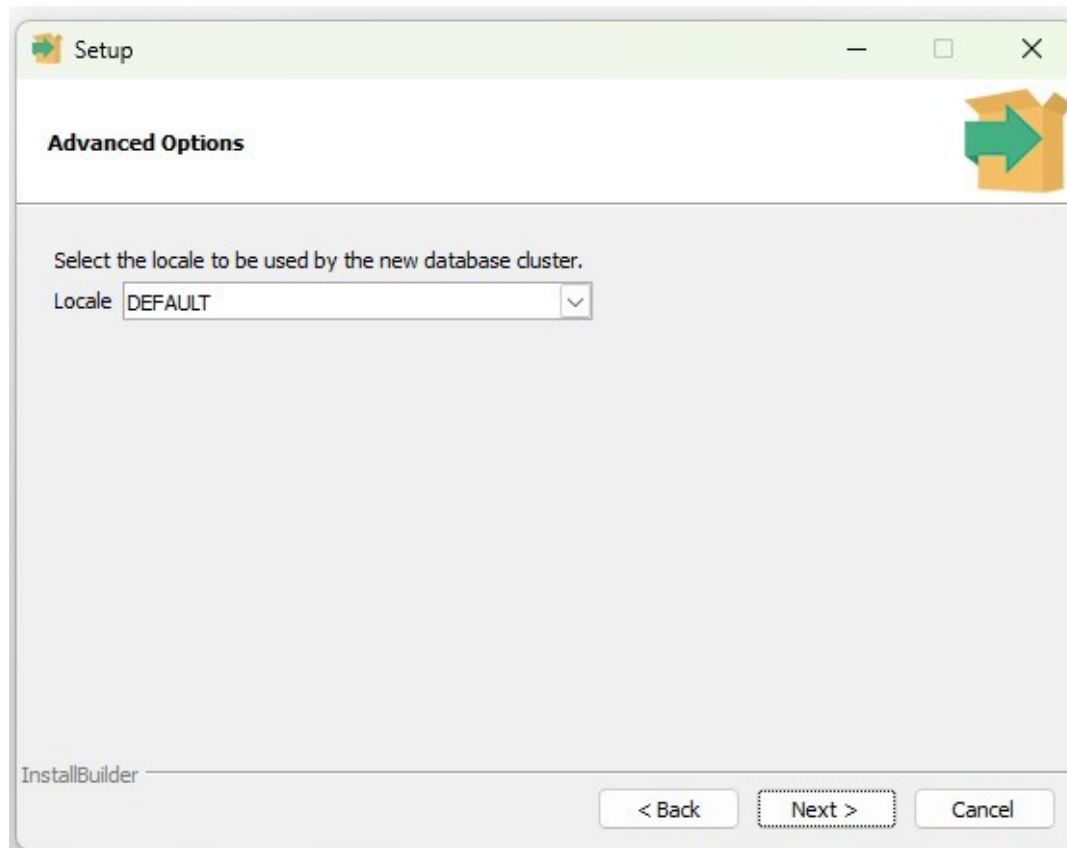


The screenshot shows a Windows-style window titled "Setup" with a green arrow icon in the top-left corner. The window has a title bar with standard minimize, maximize, and close buttons. The main content area is titled "Password" and contains the instruction: "Please provide a password for the database superuser (postgres)." Below this instruction are two text input fields. The first field is labeled "Password" and contains seven black dots. The second field is labeled "Retype password" and also contains seven black dots. In the bottom-left corner of the window, the text "InstallBuilder" is visible. In the bottom-right corner, there are three buttons: "< Back", "Next >", and "Cancel". A green arrow icon is also present in the top-right corner of the window.

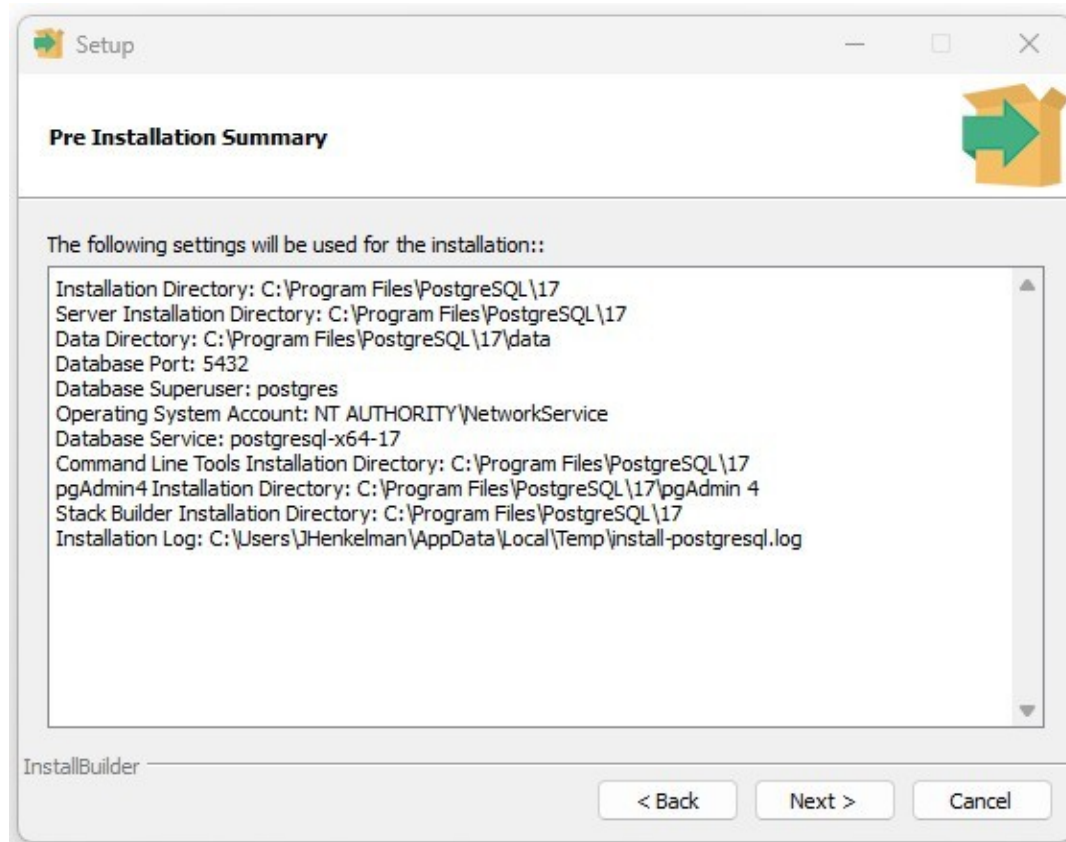
PostgreSQL Install



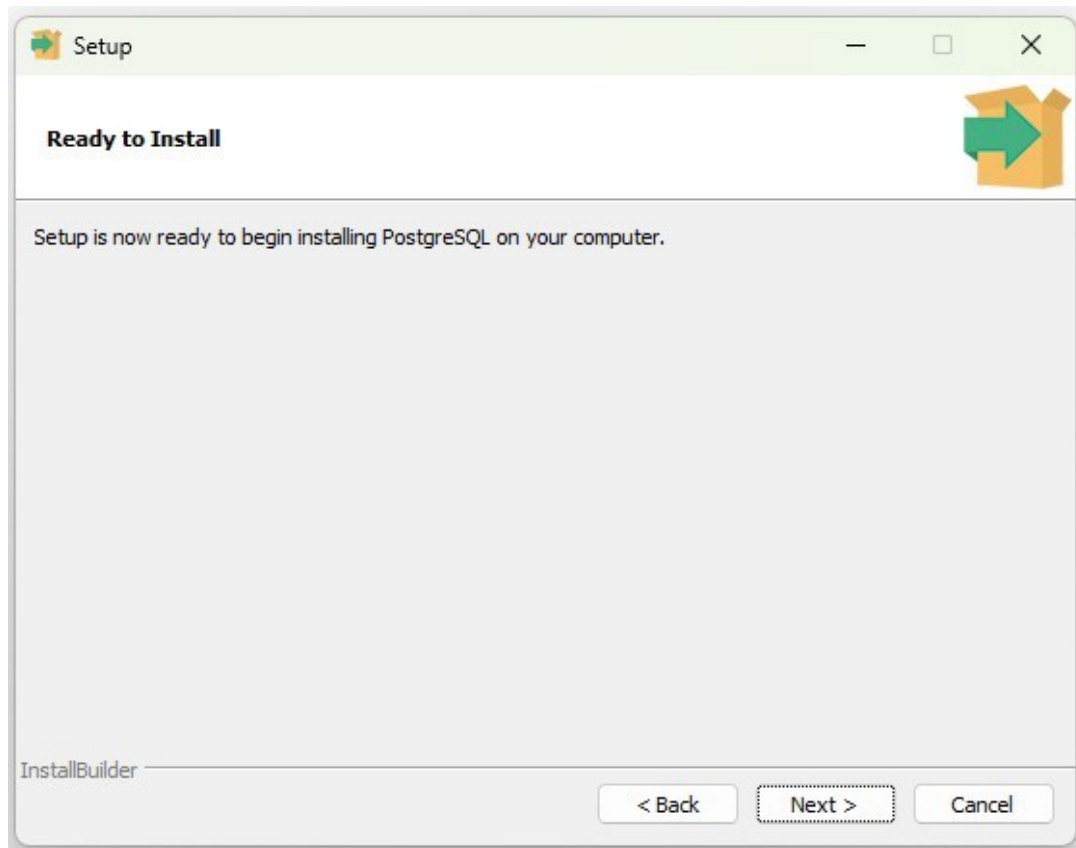
PostgreSQL Install



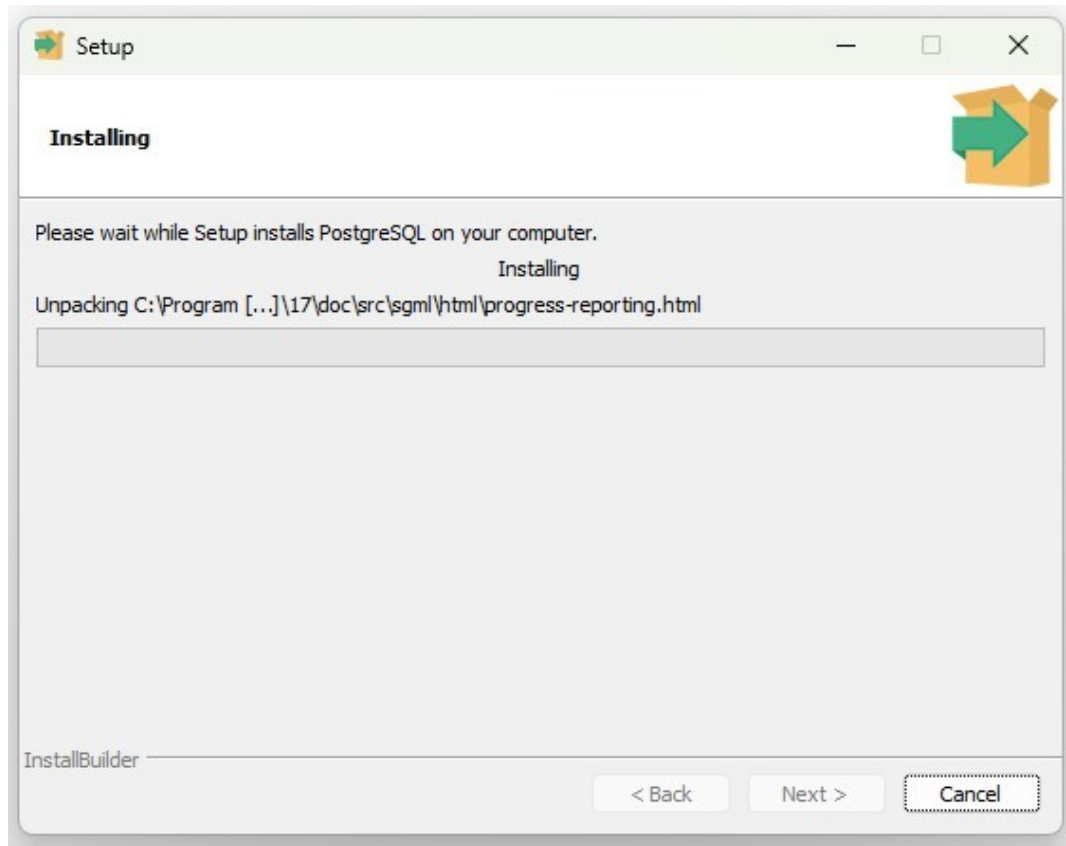
PostgreSQL Install



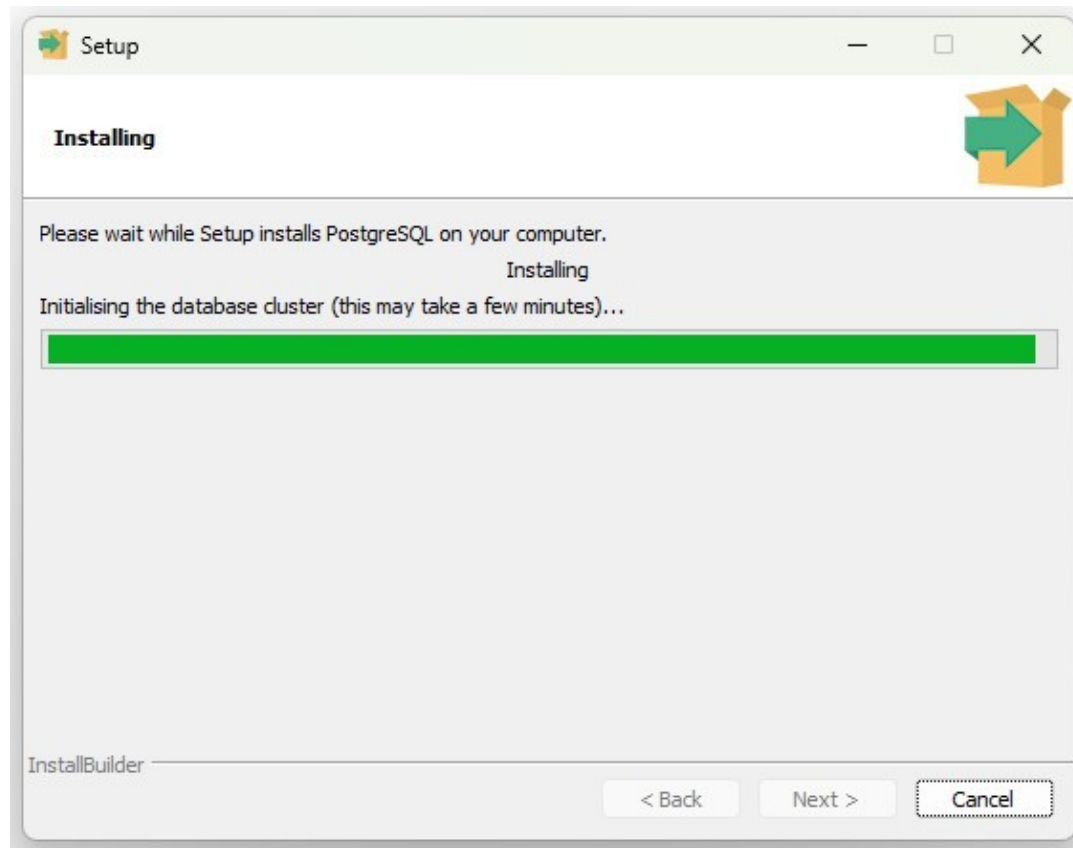
PostgreSQL Install



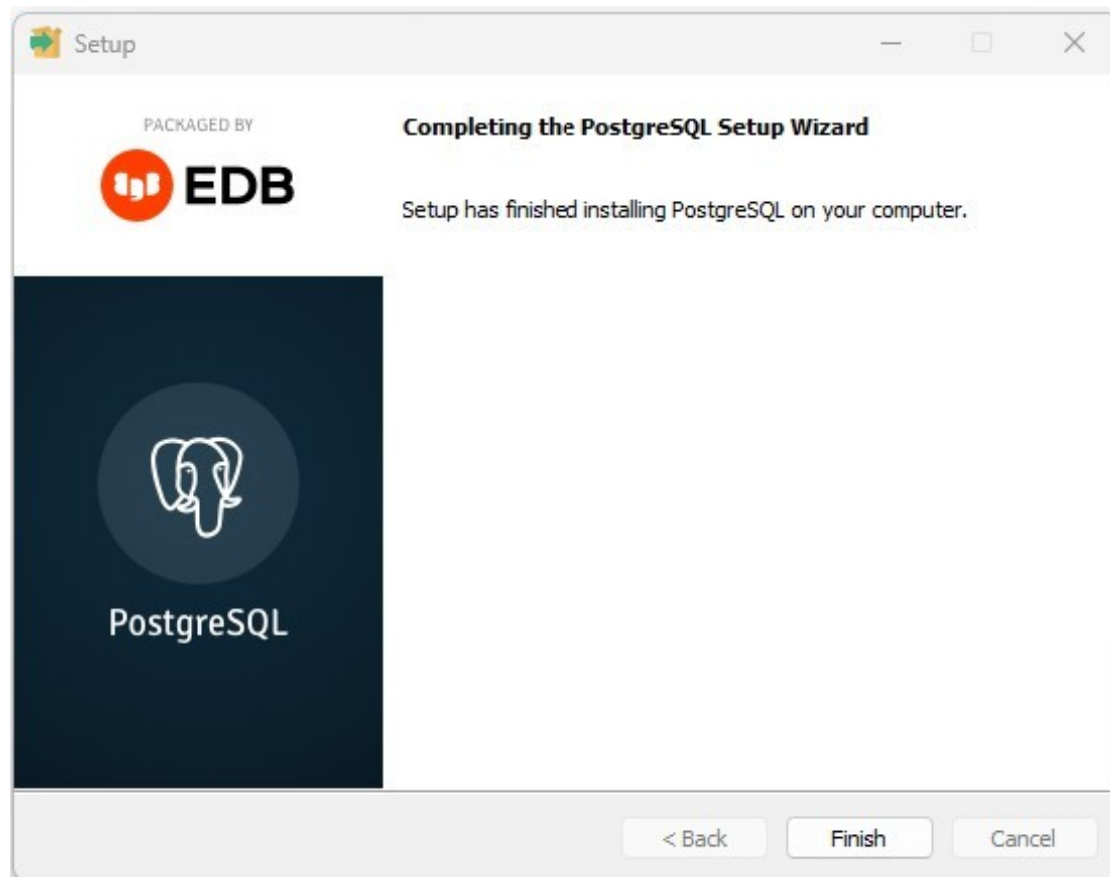
PostgreSQL Install



PostgreSQL Install

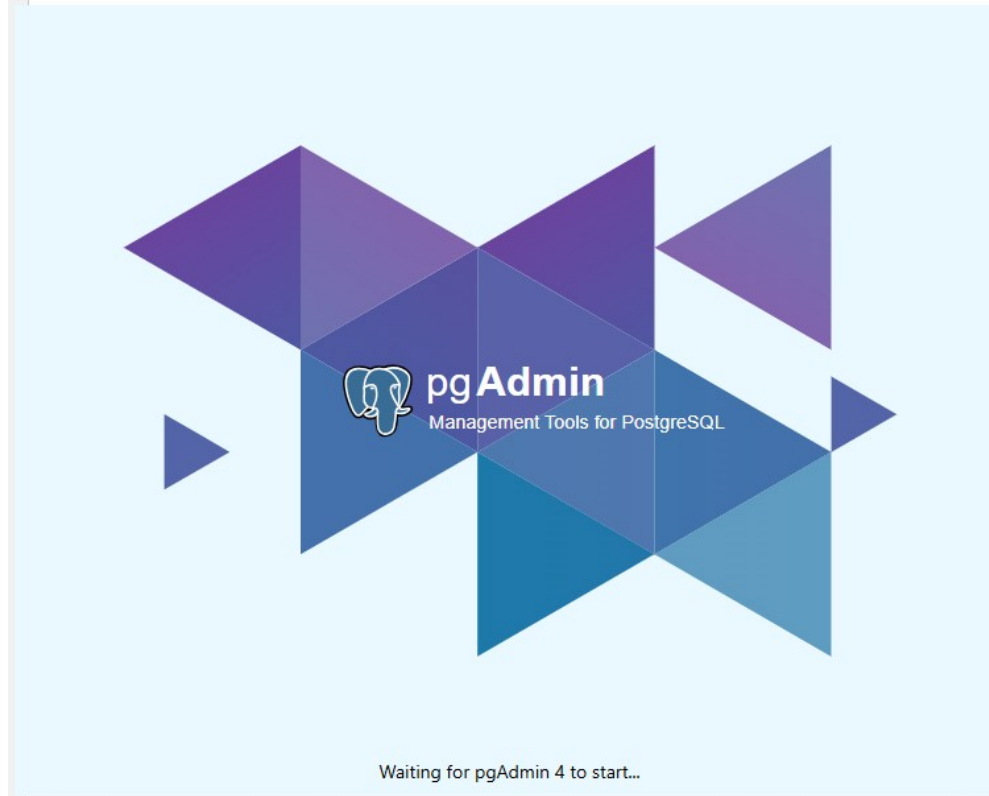


PostgreSQL Install

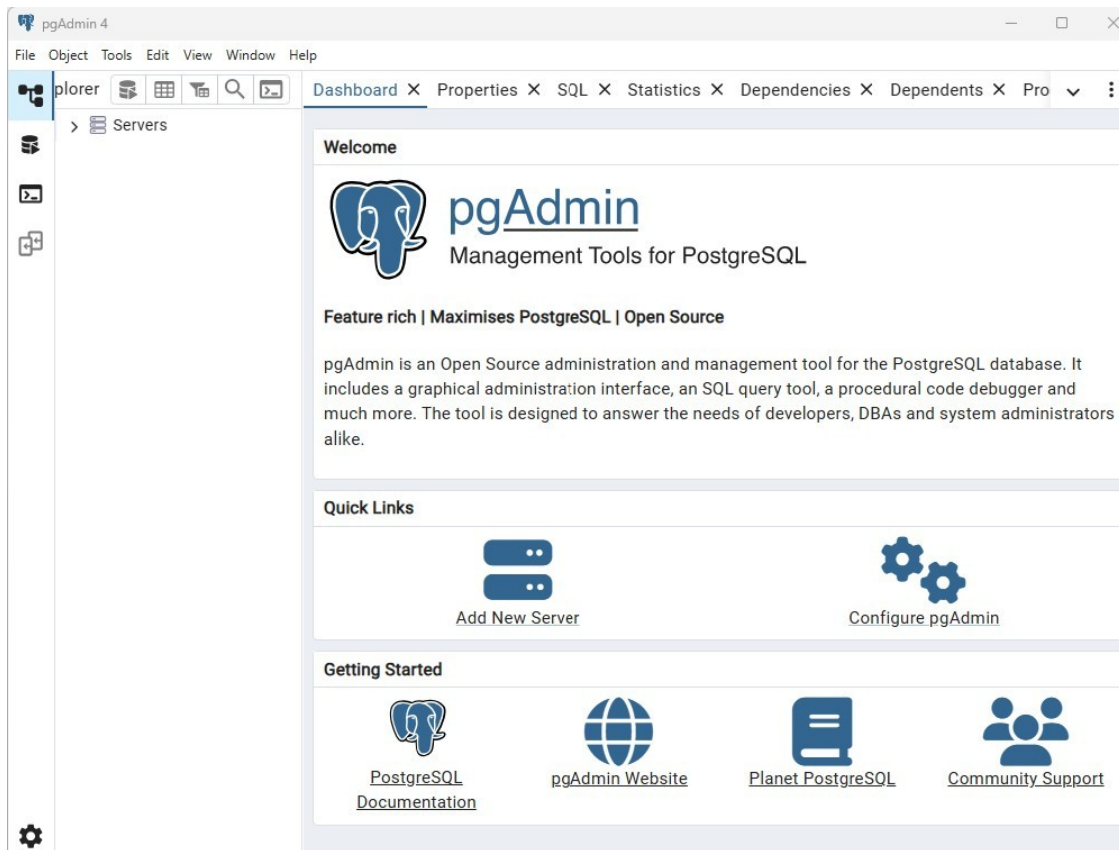


PGAdmin

Open pgadmin4

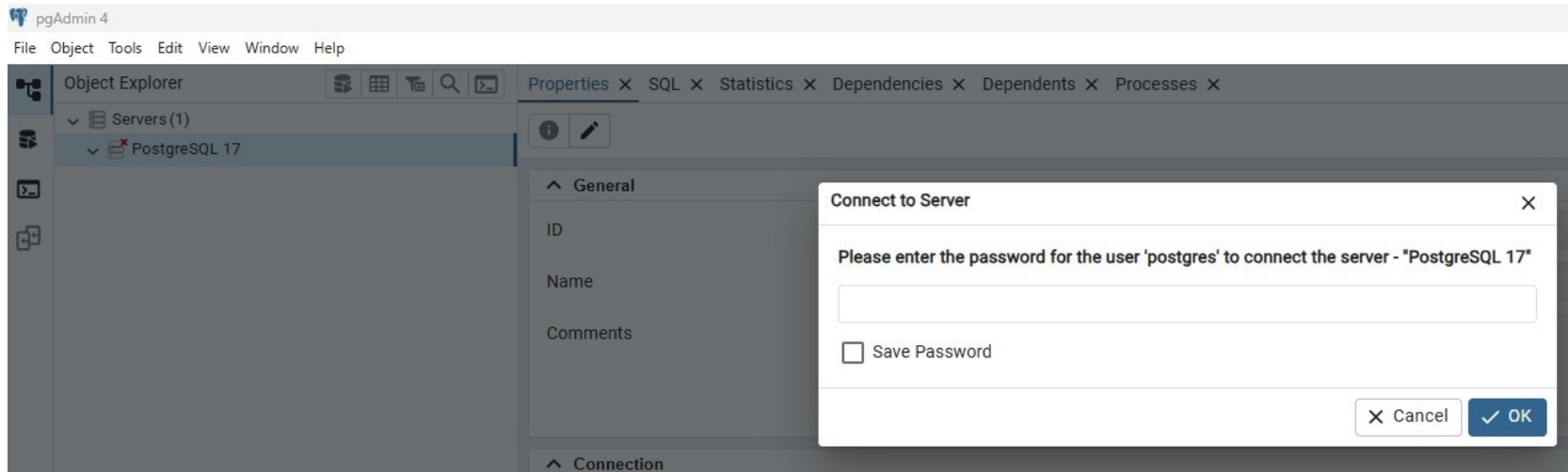


PGAdmin



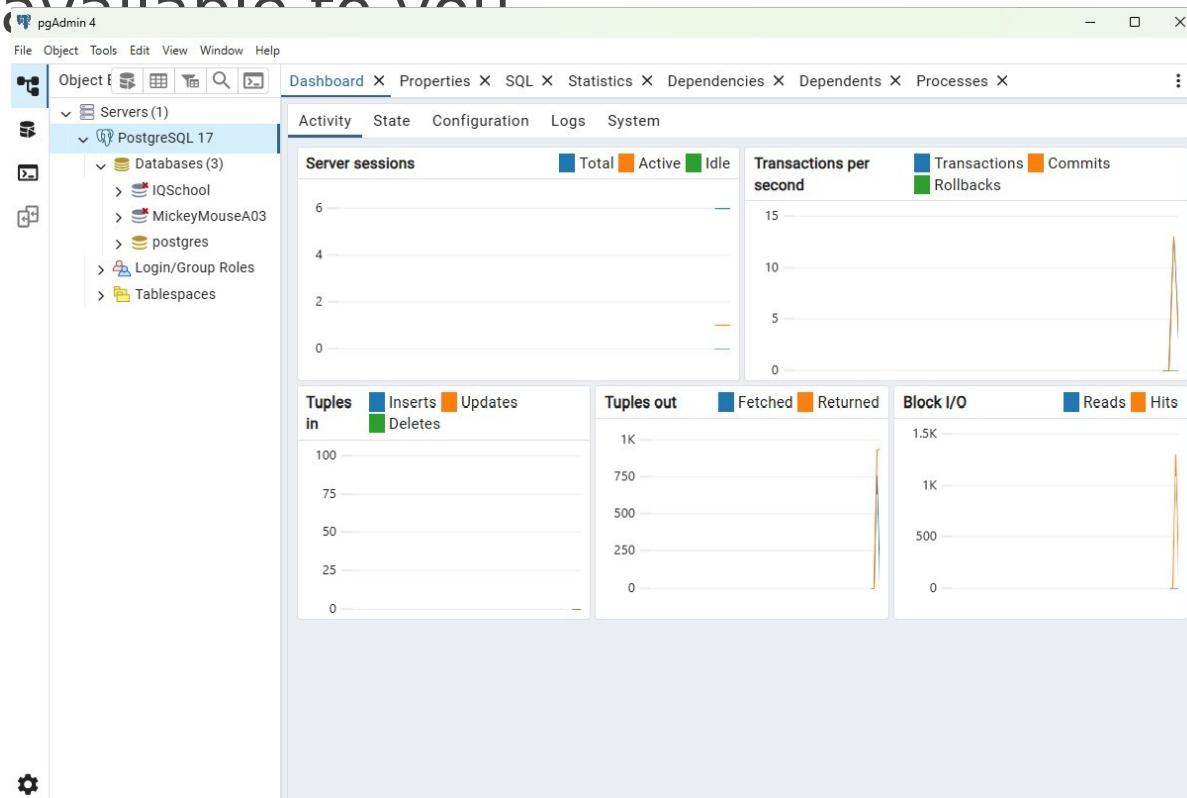
PGAdmin

When you open pgAdmin for the first time, you need to enter your server password. You can save the password or choose to enter the password each time you open pgAdmin.



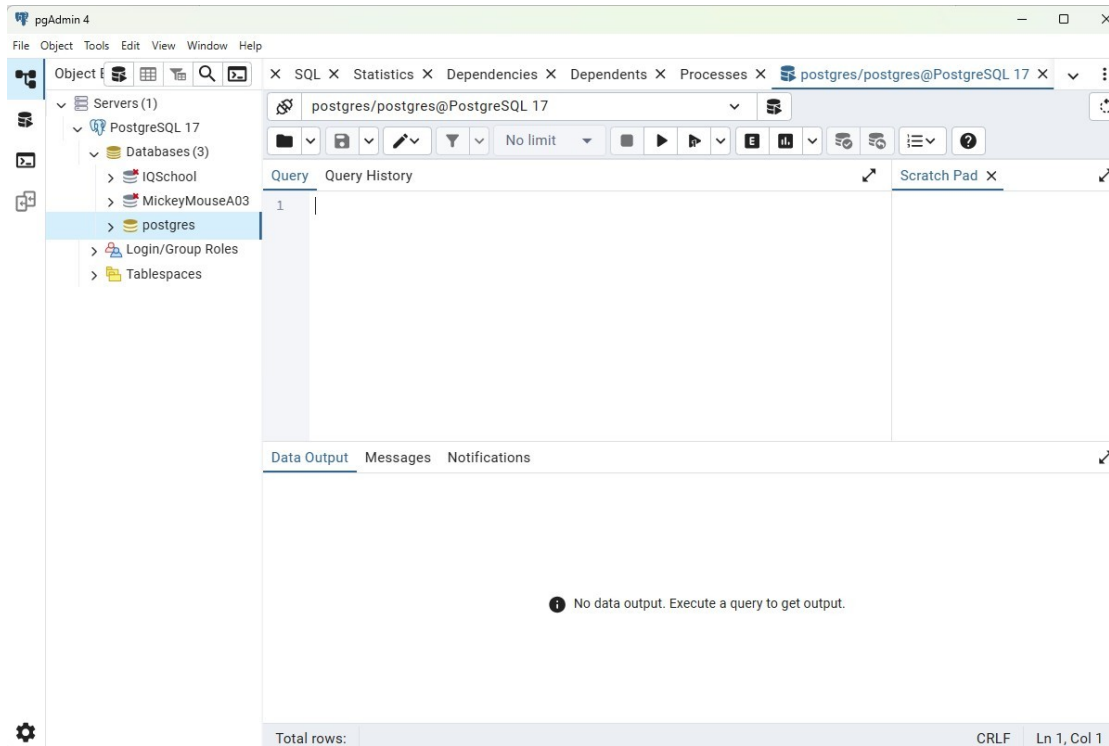
PGAdmin

Click on Servers. You should see the database you have available to you.



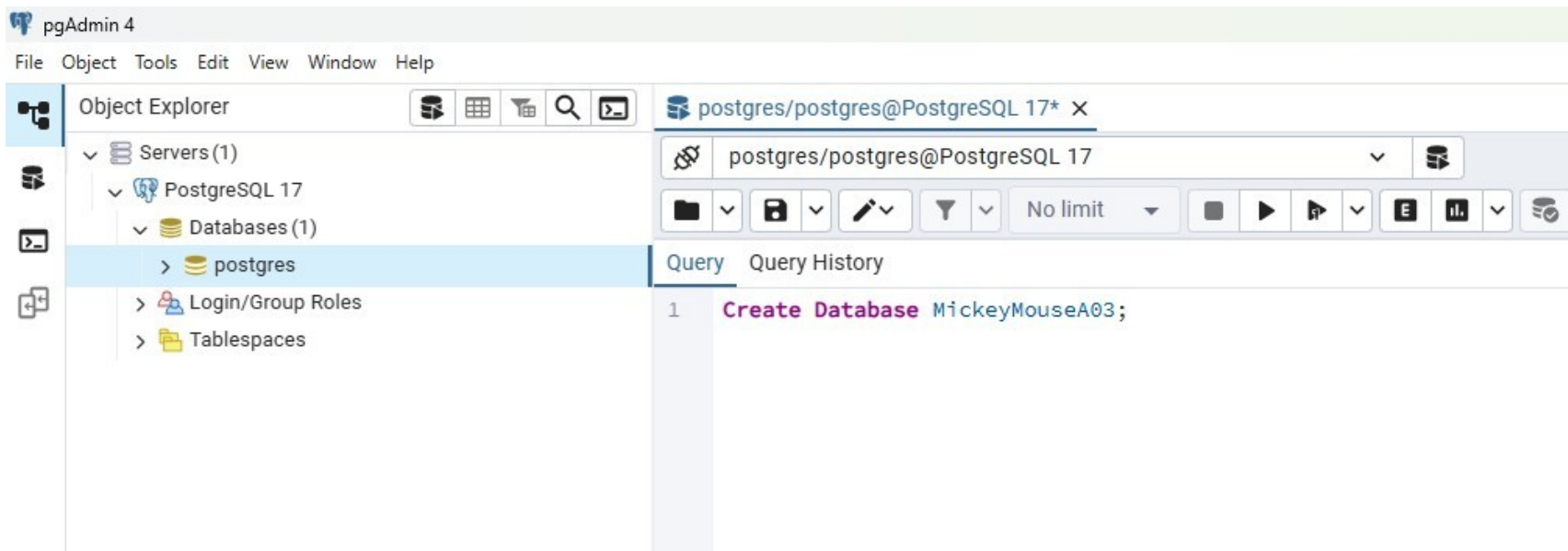
PGAdmin

Right click on the database you want to use and click Query Tool, or use Alt + Shift + Q.



PGAdmin

Type your SQL command in the Query Tool. Then press the ► arrow key or F5 key to execute that command.



Recommended

From the Brightspace site access and review the following:

- 2A Entity Relationship Model
- 2 A ERDs Definition Review
- Crow's Foot Notation

Important: ERD Symbols & Terminology quiz next Monday, September 15, 2025.